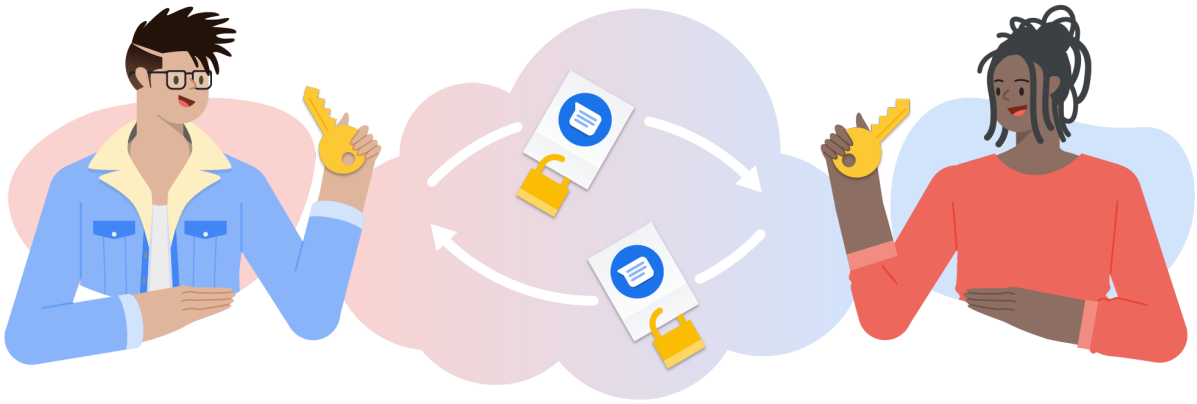
Messages End-to-End Encryption Overview

Technical Paper

June 2021

Version 1.1



A high-level technical overview of end-to-end encryption in Messages

Introduction **2**
Background 3
Threat Model 3
Goals 4
UI Changes **4**
SMS/MMS Fallback 5
Identity Verification 6
E2EE in Messages **6**
Signal Protocol 6
Key Server 7
Messages Encryption 8
Attachment Encryption 9
Session Recovery 9
Web Client 9
RCS Servers 10
Storage & Access 10
Android Messages Database 10
Notifications 10
Wear OS 10
Limitations **11**
Third Party RCS Client 11
Conclusion **12**

Introduction

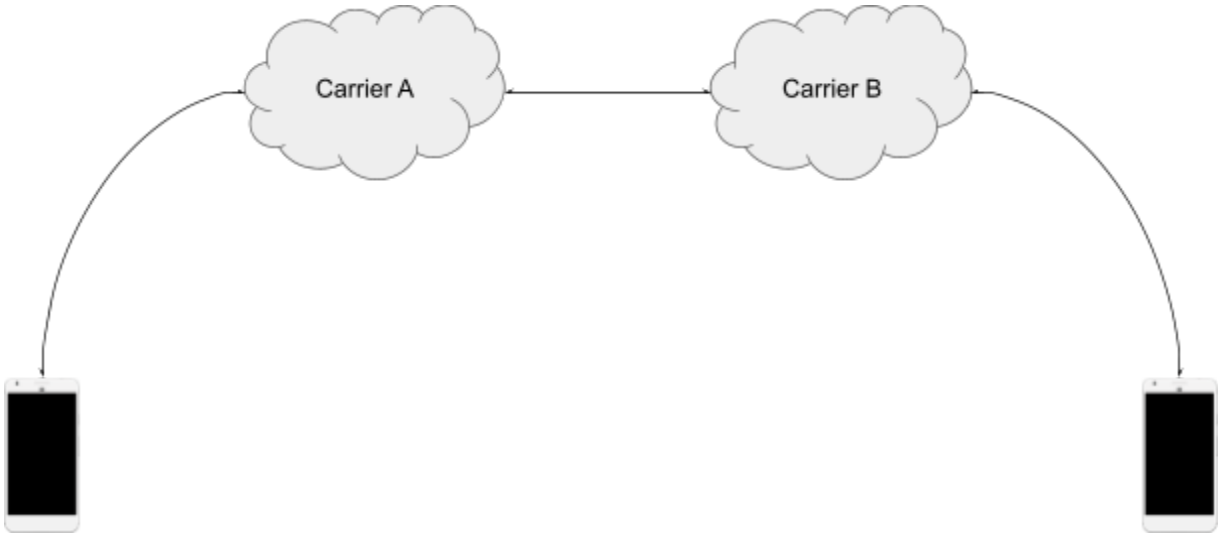
Rich Communication Services (RCS) is designed to improve users' experience and security over Short Message Service (SMS)/Multimedia Messaging Service (MMS), and we've invested in making

Messages by Google a modern and globally available RCS and SMS/MMS messaging app for Android phones. While RCS is already a big security improvement over SMS/MMS, we wanted to take it a step further and add end-to-end encryption (E2EE) to Messages, so no one else – including Google servers or third-party servers – can access your messages as they travel between your phone and the phone you message to.

Background

RCS uses the standard Session Initiation Protocol (SIP)[1] to establish a connection between two clients through a network of RCS messaging servers. This connection is then used to exchange messages using Message Session Relay Protocol (MSRP)[2]. Most RCS server deployments are either hosted by a carrier or by Jibe Mobile from Google.

In situations where the two RCS clients are not on the same carrier network, they're connected through multiple servers — one from each carrier. The client-to-server and server-to-server network connections are all encrypted using Transport Layer Security (TLS).



Threat Model

With end-to-end encryption, users' message content can only be accessed by the clients in the conversation. The message servers, either hosted by Google or by the carrier, are capable of routing the messages, but are not able to access the content. This helps to protect against:

- **A malicious or compromised server**: If an attacker compromises the messaging server, they shouldn't be able to view, modify, or replay the message contents. The best they can do is to access the messages' metadata or perform a denial of service (DoS) attack by dropping the messages.
- **A malicious user**: Users can only decrypt messages addressed to them. They can't impersonate other users without being detected.

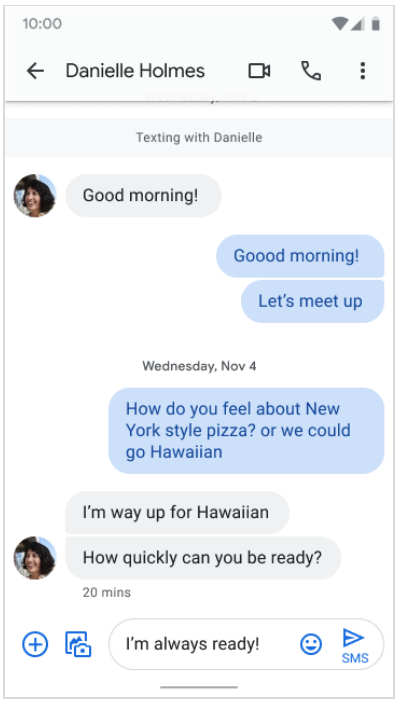
Goals

- **Confidentiality**
  Only clients in the conversation can access the message content.
- **Integrity**
  The encrypted message can't be modified in transit without being detected.
- **Authentication**
  A malicious client can't spoof another client phone number and send encrypted messages without being detected.
- **Forward Secrecy & Post-Compromise Security**
  If one message happens to be compromised, the security of all past and future messages is still maintained.
- **User Experience**
  Provide a consistently secure user experience.

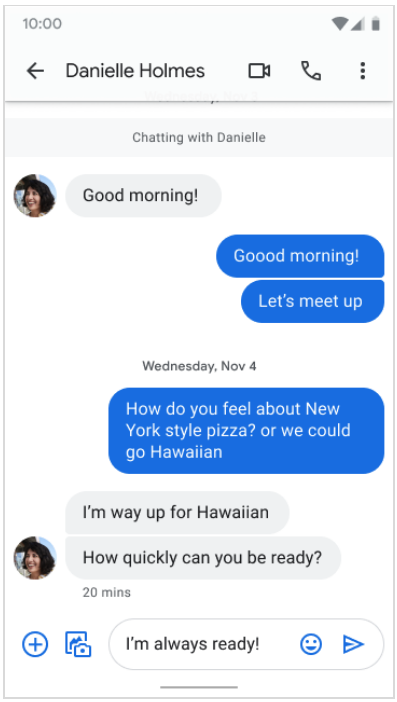
UI Changes

Starting with one-to-one conversations, all RCS chat messages will be E2EE if both clients have the latest version of Messages. Conversations already automatically upgrade from SMS to RCS when eligible, and now they will upgrade to E2EE when eligible. Messages already differentiates between RCS and SMS/MMS messages by using different color shades. To signify when the conversation is end-to-end encrypted, we added lock icons on the send button in the compose field and next to the timestamp of the message.

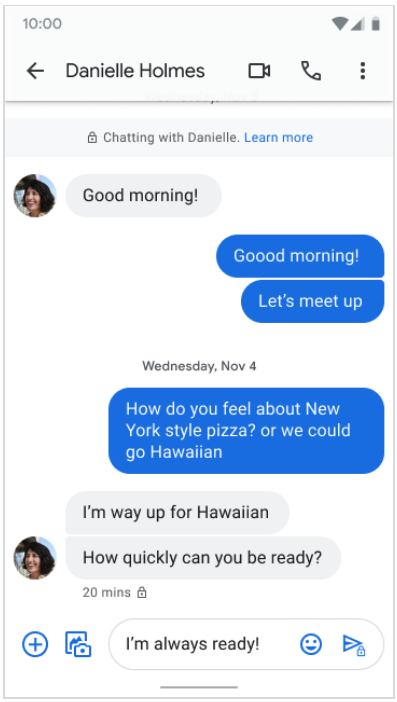
SMS/MMS



RCS "Chat"

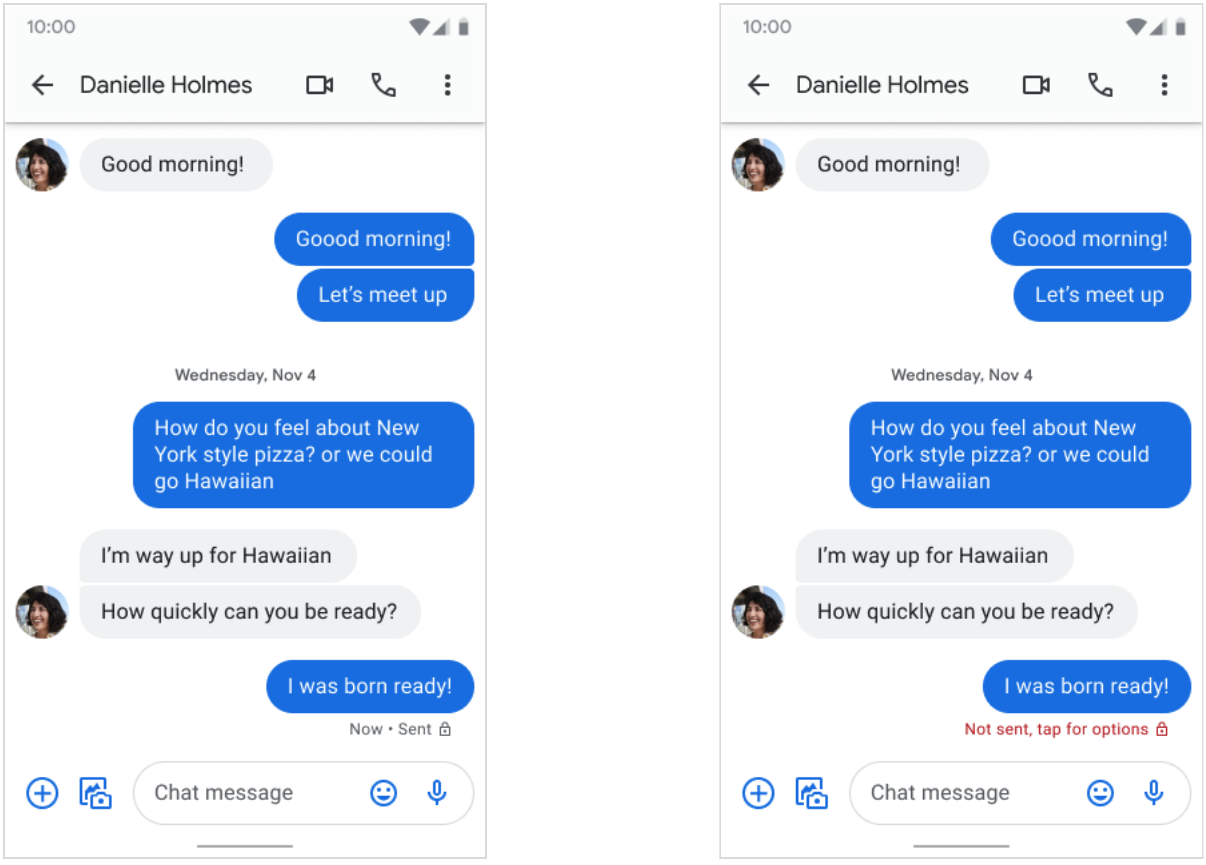


E2EE RCS "Chat"



SMS/MMS Fallback

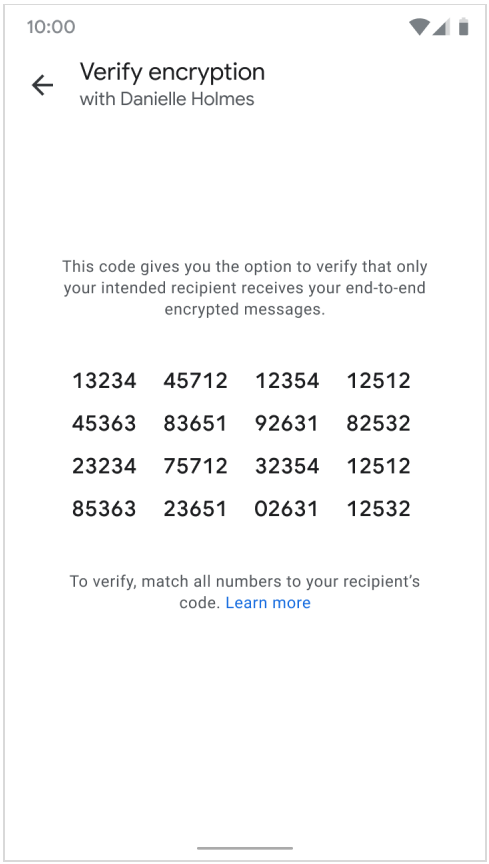
One of the many benefits of default messaging apps is that users can connect over cellular or data networks. With RCS providing better experience and privacy than SMS/MMS, we have updated the fallback behavior so messages, when eligible, will send over RCS and not automatically fall back to SMS/MMS. If the recipient is offline, these messages will be stored in the server and delivered when the recipient is back online, but the sender can choose to resend unsent RCS messages as SMS/MMS instead.



A sent end-to-end encrypted message (left) and if the message fails to send (right).

Identity Verification

For additional assurance that the conversation is E2EE, users can verify they are getting the right public key for the other side of the conversation by comparing the verification code, which is the fingerprint of the two identity keys. This step is only needed when users message for the first time or receive a new identity key due to app reinstall or switching to a new device.



The fingerprint is a numeric string generated by doing 512 iterations of SHA512 on a concatenation of both clients' identity keys.

E2EE in Messages

The following section will cover the technical details for how Messages implemented E2EE for RCS, which is currently available for one-to-one RCS conversations between the latest version of Messages.

Signal Protocol

Messages uses the Signal Protocol[3] to build E2EE for RCS messages, which allows us to achieve all of the security properties mentioned earlier.

When the client is set up, it generates the following Curve25519 key pairs:

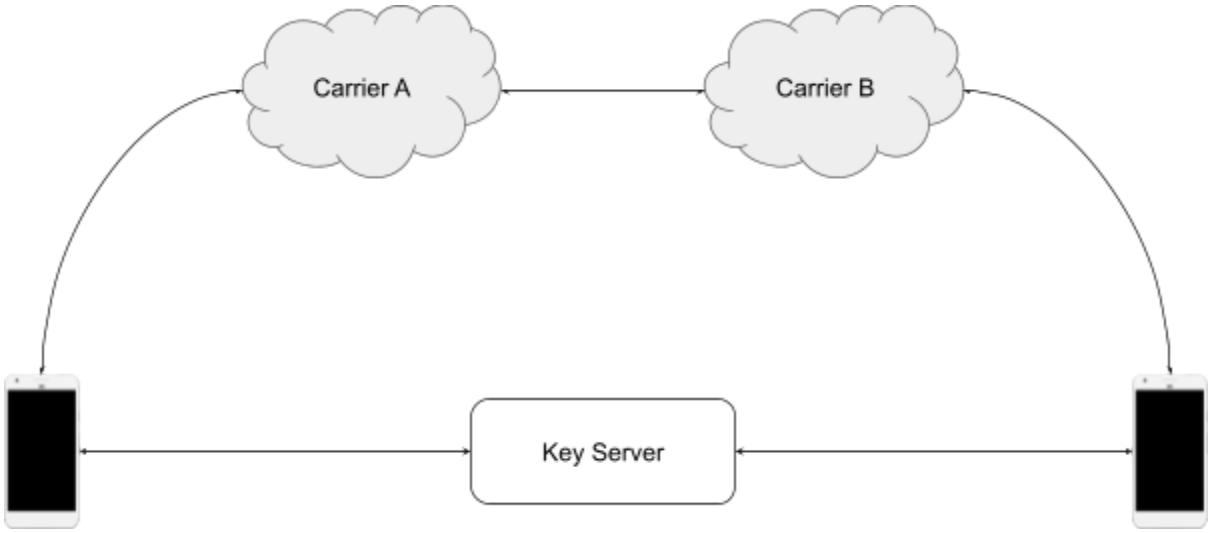
- **Identity key**: A long-term key pair associated with the user on this device
- **Signed prekey**: A key pair, with the public key signed by the Identity Key
- **Unsigned prekeys**: A set of disposable one-time use prekeys pairs

These keys are generated using the BoringSSL RAND_bytes secure random function. The public keys of these keys are uploaded to a Google key server, while the private keys never leave the device.

Signal uses these keys to set up the E2EE session between two clients using the triple Diffie-Hellman algorithm (X3DH)[4] and derives 256-bit root key material from them. The root key is used to derive another 256-bit key material called chain key. From the chain key, it derives the message encryption key, authentication key, and initialization vector. Every message is encrypted with different keys as the chain key and root keys change using the Double-Ratcheting[5] algorithm.

Key Server

In order to store and exchange user public keys like identity keys and prekeys, we need to have a central key server. Unlike the RCS messaging servers, the key server is currently only hosted by Google.



When the client turns RCS on for the first time, it also registers with the key server and uploads the public key parts that are used for session setup. For example, when Alice talks to Bob for the first time, her client requests the following information from the key server:

- Bob's identity key
- Bob's signed prekey
- One of Bob's prekeys

The server deletes the returned prekey from its storage. Alice uses these keys in the triple Diffie-Hellman algorithm to derive the session keys.

When the key server detects that one of the clients is low on prekeys, it sends a push notification for that client to generate and upload a new set of prekeys and a new signed prekey.

Messages Encryption

Once a secure session is established using remote client prekeys, Signal derives the following values:

- 256-bit AES encryption key
- 256-bit MAC key
- 128-bit initialization vector

Signal uses AES-256-CBC with PKCS#7 padding for message encryption. The encrypted message is stored in a protocol buffer along with other session states. A 64-bit MAC is computed over the serialized protocol buffer using HMAC-SHA256 to create the final message payload. After each encryption, Signal advances the chain key to achieve forward secrecy.

The encrypted payload is encoded in Base64 and added to the message body while keeping everything else, like sender & recipient info and timestamps, in plaintext so the messaging servers can properly route the messages.

```
content-type: message/cpim
...
To: +12075550101
From: +12075550102
DateTime: 2020-11-18T18:13:07.861Z

Content-Length: 215
Content-Type: application/vnd.google.rcs.encrypted; charset=utf-8

CnYIARJyMwohBdeTpErO+LvfI9dvQt0IJlm44sd+6bmIBa4ca4kjNj4KEAAYACJA5d1VbsJgz2QCy
bdpzVml0pMWVzfYXCJZu/KbuRidycuHMfQ3pAbalGNQeZfVo7EpG3GtEKeXJypXLtRSeolSNLtRUg
akGL7PEiQwZjRmYmYzMy1mYzdkLTRmOWMtYTA0MC00NjQyNDE3MjExY2Y=
```

The RCS specification defines several types of messages. Our implementation of E2EE uses varying strategies for encrypting each type of message to maximize user privacy while still adhering to the RCS specification.

For most user-generated messages (text, location, reactions) and read receipts, we've introduced a new "rcs.encrypted" content type. This format encrypts the entire message payload, preventing all messaging servers from knowing the type or being able to access the content (see figure above).

Typing indicator messages and delivery receipts use standard RCS format for RCS operational reasons, making metadata such as message IDs of delivered messages in the receipts visible to the servers. In the case of delivery receipts, we have extended the XML with an additional encrypted element that is used by the client to authenticate the receipt.

Attachment Encryption

Attachments in one-to-one conversations are also secured by E2EE. In standard RCS, an attachment is uploaded to the sending user's carrier content store, then a link to that file along with other metadata, such as filename and size, are sent to the recipient in the RCS message in an XML payload. With E2EE, we divided this into steps:

1. **File encryption**: Before the sender client uploads the file, it generates a fresh 256-bit random key material from which it derives a 256-bit symmetric encryption key and a 128-bit initialization vector which are used to encrypt the file using AES-CTR 256. Then it generates a digest using SHA256. The encrypted file is then uploaded to the carrier content store.
2. **Message encryption**: Key material, digest, and some metadata are encrypted using the Signal session. The RCS file transfer XML is extended to include this encrypted payload. Private metadata such as file names and content types are only transmitted in the encrypted payload. Metadata such as size and URL to the encrypted file are directly available in the RCS file transfer XML for the RCS servers to operate some of its functionalities.

The recipient client:

1. Decrypts the encrypted payload in the message to retrieve the file metadata, key material, and message digest.
2. Downloads the encrypted file from the content store.
3. Verifies the digest, dropping the message if verification fails.
4. Derives the encryption key and IV from the key material.
5. Decrypts the file.

Session Recovery

If the recipient client loses its local encryption session state for any reason, such as app reinstall or switching to a new phone, and receives a message encrypted using the old session keys, the recipient client will fail to decrypt that message. The recipient sends an error control message to the sender which automatically reestablishes the session by requesting a new prekey from the keys server, then re-encrypts and sends the message.

Web Client

Messages for web also supports E2EE conversations by using the phone to send and receive messages. The client establishes a secure connection with the phone by scanning a shared key encoded in a QR code. The QR code contains a 256-bit symmetric encryption key and a 256-bit authentication key.

Messages between the web client and phone are encrypted using AES256-CTR with a random 128-bit initialization vector and authenticated using HMAC-SHA256. The Google servers relaying the data can't access the contents sent between the web client and the phone.

RCS Servers

Like regular RCS messages, E2EE RCS messages are delivered through RCS servers that are operated by carriers and Google. E2EE makes message content invisible to servers and parties outside of the conversation, but certain operational or protocol metadata can still be accessed and used by the servers, including:

- Phone numbers of senders and recipients
- Timestamps of the messages
- IP addresses or other connection information
- Sender and recipient's mobile carriers
- SIP, MSRP, or CPIM headers, such as *User-Agent* strings which may contain device manufacturers and models
- Whether the message has an attachment
- The URL on content server where the attachment is stored
- Approximated size of messages, or exact size of attachments

Storage & Access

Android Messages Database

As part of an open, interoperable ecosystem, SMS and RCS messages are stored in Android's local on-device messages database[6] to allow seamless integration with companion applications such as reading messages aloud in your car. Only applications with SMS permissions can access this database, and users can manage SMS permissions from the Android settings. Messages also uses this central database to include SMS and RCS messages in Android system backup, so messages can be transferred to a new client or device. Starting from Android version P, the Android system backup is end-to-end encrypted with a secret key derived from the user's lock screen PIN/pattern/passcode[7] so Google servers can't access it.

To keep the same user experience and avoid losing messages when users move to a new device, E2EE RCS messages will continue to be stored in the device's messages database and accessible to apps with SMS permissions. We will work across the ecosystem to improve security of E2EE storage and access by other apps without compromising the companion messaging experience.

Notifications

In addition to SMS permissions, other apps may have access to E2EE messages through incoming message notifications. This type of access can be managed by the user on a per-app basis through SMS permissions or notification access.

Wear OS

Messages support conversations sync between the phone and wear clients, which can happen via Bluetooth or through Google Cloud Sync, if enabled. Data transmitted over Cloud Sync supports encryption in transit and encryption at rest but does not yet support E2EE. E2EE via Cloud Sync will be available later this year. Users can turn off Cloud Sync in the Wear OS companion app on the phone.

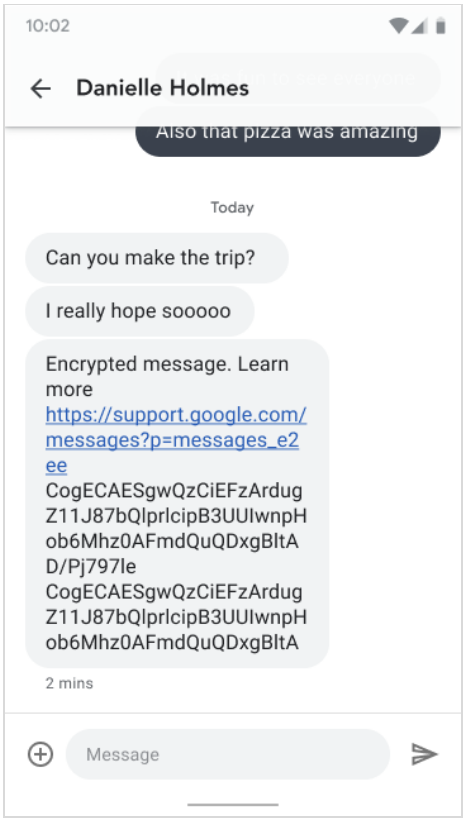
Limitations

Unlike other messaging apps where the service provider controls both client and server, RCS is a federated protocol implemented by a variety of client and server vendors. We made sure that E2EE messages will continue to be delivered though other RCS servers without breaking the user experience, but this comes with some challenges and limitations that we will continue working to overcome.

Third Party RCS Client

E2EE is implemented in the Messages client, so both clients in a conversation must use Messages, otherwise the conversation becomes unencrypted RCS. In rare situations where the conversation starts as E2EE, then one of the clients migrates to a different RCS client or an older Messages client that does not support E2EE, Messages might be unable to detect the change immediately. If the Messages user sends a new message, it's still E2EE, however the recipient client may render the encrypted base64 payload directly as message content.

To improve this situation, we append "Encrypted message..." prefix to the encrypted payload and link to the documentation to explain what is happening to the recipient user. Also, this situation should only happen for the first message, as the Messages client recovers when it receives a non-E2EE delivery receipt and downgrades the conversation to unencrypted RCS. Retrying the message at this point will use standard unencrypted RCS.



Conclusion

E2EE in Messages has been a big challenge given how complex the RCS ecosystem is. We managed to build solutions for most of these challenges and we will continue to build solutions for the rest of them while also bringing E2EE to more features in Messages. This will improve our users' privacy and security.

REFERENCES

1. Session Initiation Protocol [here]
2. Message Session Relay Protocol [here]
3. Signal Protocol [here]
4. X3DH [here]
5. Double Ratcheting [here]
6. Android Messages Database [here]
7. Android E2EE backup [here]